Binary Search Trees

7/16/2009

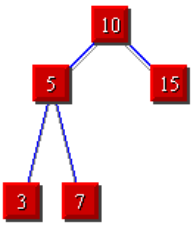
Important Dates

- **Project 2 due**
  - **DESIGN, DESIGN, DESIGN!!!**
  - **You may have 0 or 1 partner.**
    - **NO EXCEPTIONS!**
  - **Friday 7/24/2009 – 10pm**
- Project 2 – time to work in lab
  - Monday 7/20/2009
- Midterm Review
  - Saturday 7/26/2009 – 1-4pm in 306 Soda
- Midterm 2
  - Tuesday 7/28/2009 – 5-6pm in 10 Evans

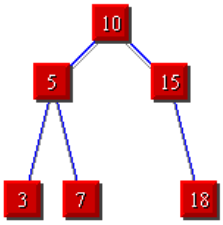
Topics

- Maximally balanced trees
- Big-Oh of tree operations
- Binary Search Tree remove
- Stacks
- Map

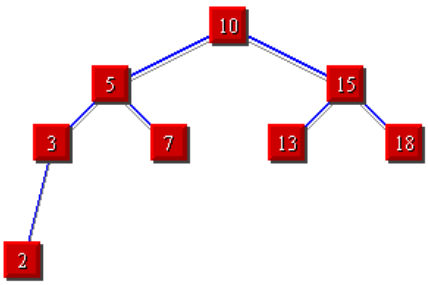
Maximally Balanced?



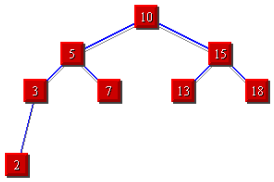
Maximally Balanced?



Maximally Balanced?

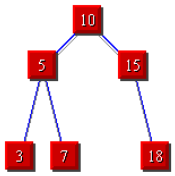


Another way to implement trees!



10	5	15	3	7	13	18	2		

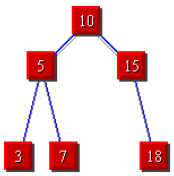
Maximally Balanced Trees – WHY?



10	5	15	3	7		18			

A Better Tree Implementation

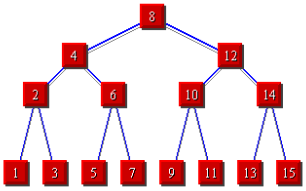
(don't use the 0 index)



Left Child at 2n
Right Child at 2n + 1

	10	5	15	3	7		18			
0	1	2	3	4	5	6	7	8	9	10

How many elements are in a balanced tree? Assume h = height



	8	4	12	2	6	10	14	1	3	5	7	9	11	13	15
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$$n = 2^{h+1} - 1$$

$$n = 2^{h+1} - 1$$

// Take the log of both sides

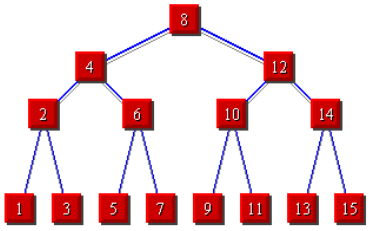
$$log_2(n) = log_2(2^{h+1} - 1)$$

// Remember:

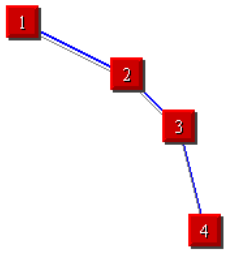
$$log_x z = y \qquad x^y = z$$

$$log_2 2^{h-1} = h - 1 \qquad \boxed{log_2 n \approx h}$$

What is the run-time for find in this tree?

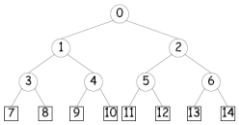


What is the run-time for find in this tree?



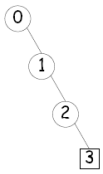
Running Times

- In a perfectly (full) balanced binary tree with height/depth *h*, the number of nodes $n = 2^{(h+1)} - 1$.
- Therefore, no node has depth greater than $log_2 n$.
- The running times of `find()`, `insert()`, and `remove()` are all proportional to the depth of the last node encountered, so they all run in *O(log n)* worst-case time on a perfectly balanced tree.



Running Times

- What's the running time for this binary tree?
- The running times of `find()`, `insert()`, and `remove()` are all proportional to the depth of the last node encountered, but $d = n - 1$, so they all run in *O(n)* worst-case time.



Running Times

- The Middle ground: reasonably well-balanced binary trees
  - Search tree operations will run in *O(log n)* time.
- You may need to resort to experiment to determine whether any particular application will use binary search trees in a way that tends to generate balanced trees or not.

Running Times

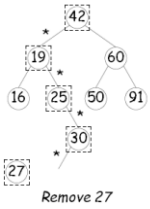
- Binary search trees offer *O(log n)* performance on insertions of randomly chosen or randomly ordered keys (with high probability).
- Technically, all operations on binary search trees have *Theta(n)* worst-case running time.
- Algorithms exists for keeping search trees balanced. e.g.,2-3-4 trees.

Binary Tree Remove

http://www.cs.jhu.edu/~goodrich/dsa/trees/btree.html

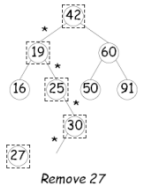
Delete a node given a key, if a node exists.

1. Find a node with key *k* using the same algorithm as `find()`.
2. Return `null` if *k* is not in the tree;
3. Otherwise, let *n* be the first node with key *k*. If *n* has no children, detach it from its parent and throw it away.

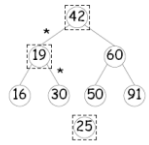


Binary Tree Remove

4. If *n* has one child, move *n*'s child up to take *n*'s place. *n*'s parent becomes the parent of *n*'s child, and *n*'s child becomes the child of *n's* parent. Dispose of *n*.



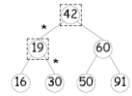
Remove 27

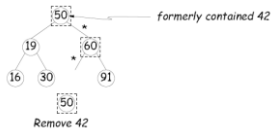


Remove 25

Deletion

5. If *n* has two children:
   - Let *x* be the node in *n*'s ***right subtree*** with the ***smallest*** key. (the inorder successor) Remove *x*; since *x* has the ***minimum*** key in the subtree, *x* has ***no left child*** and is easily removed.
   - Replace *n*'s entry with *x*'s entry. *x* has the closest key to *k* that isn't smaller than *k*, so thebinary search tree invariant still holds.



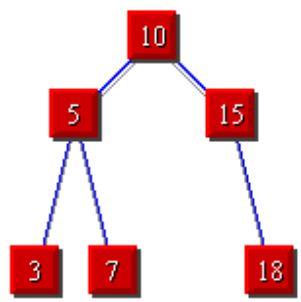


Remove 42

Stack

- `push` : put something on top of the stack
- `pop`: take something from the top of the stack
- `peak`: look at the thing on the top of the stack

Stack Based Pre-Order Traversal



Map is an Interface

KEY	VALUE
Jonathan	Hot pink
Kaushik	Green
David	Blue
George	Green
Colleen	Yellow

Map Interface

- `get` : returns the value for that key
- `put`: takes a key and a value and adds them to the map.
- `containsKey` : returns a boolean